22 Programming the keys

"Pride leads only to shame."

22.1 Secrets

22.1.1 Keyboard controller

Normally nobody uses PC/XT (8bit) systems, we use AT (16/32/64bit) systems. So I think it is enough to explain the secrets of keyboard in AT systems.

In a typical AT system, the microcontroller(8048,6805 type) in the keyboard sends data to the keyboard-controller (8042 type) on the motherboard. Controller found on the motherboard can also send data back to the keyboard.

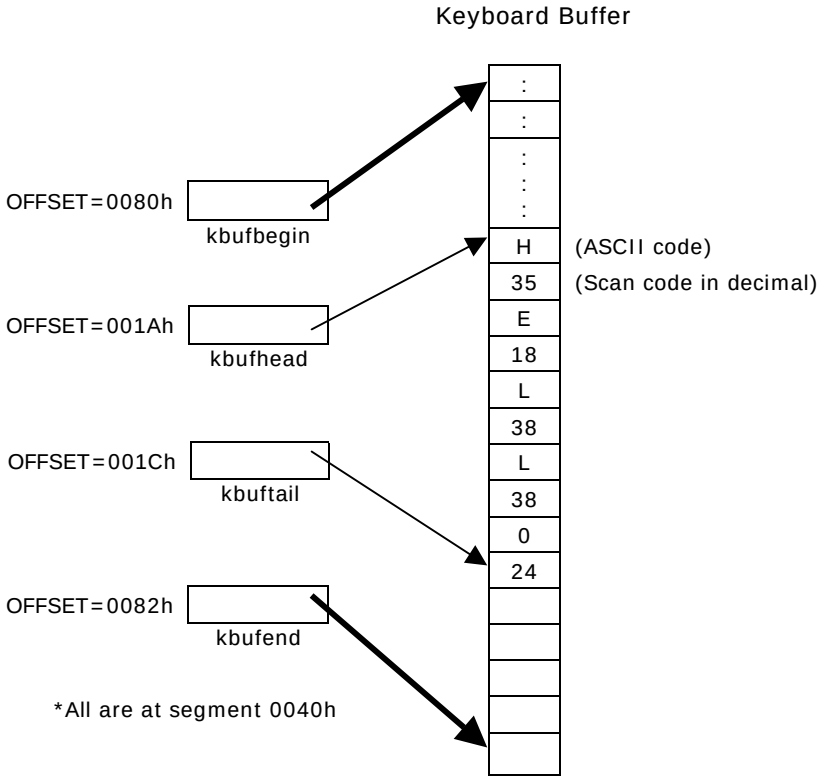
In detail, a keyboard consists of set of switches mounted in a grid (key matrix). When you press a key on the keyboard the micro controller in keyboard reads the key switch location in the key matrix, then it sends data to keyboard-controller on the motherboard. When the keyboard-controller on the motherboard receives data, it signals the motherboard with an IRQ1 and sends data to the main motherboard processor via I/O port address 60h. The function of the keyboard-controller on the motherboard is to translate scan codes and perform other functions. We can use I/O port 64h(R/W) to check the status of the keyboard-controller on the motherboard.

Note
Some people call the keyboard-controller on the motherboard as *keyboard BIOS*

Note
Scan code is different from ASCII code. The upper and lower case is determined by the state of shift keys, not solely by which key is pressed

22.1.2 Keyboard Buffer

A part of the PC's BIOS data area i.e., memory at segment 0040h is used as keyboard buffer. This area also holds pointers to keyboard buffer and key status.



The keyboard buffer is organized as a circular queue. It has four 2-byte wide pointers: `kbufbegin`, `kbufend`, `kbufhead` and `kbuftail`. Here you should note one important thing: these pointers are just 2-byte wide (not 4-byte wide), which means these pointers hold only the OFFSET address (all are at segment 0040h). `kbufbegin` and `kbufend` points to the beginning and end of the keyboard buffer and these pointers do not move. Whereas the `kbufhead` and `kbuftail` points to the character on the keyboard buffer and so these pointers do move.

Keyboard buffer is a character (i.e., 1 byte wide) array. The size of the keyboard buffer may vary from system to system. Some people say that the size of the keyboard buffer is 32 bytes, which is wrong, because the size of the keyboard buffer can be changed. Keyboard buffer holds ASCII code and scan code on alternate bytes.

Whenever a key is been inputted through keyboard, it is being temporarily stored in keyboard buffer, before it is processed by the BIOS. When we try to input more keystrokes, we will get a beep sound indicating that the keyboard buffer is full. The pointer `kbuftail` points to the recently inputted key and the pointer `kbufhead` points to the key that is being currently processed. So when the keyboard buffer is empty, the pointer `kbufhead` and `kbuftail` holds the same address (i.e., points to the same data).

22.1.3 Keyboard status

The status of the keyboard i.e., whether CAPS LOCK is ON or OFF can be set with our program. For that we have two ways.

22.1.3.1 Changing keyboard status with BIOS handler

```
#include <dos.h>

#define ON  (1)
#define OFF (0)

#define SCROLLLOCK (1 << 4)
#define NUMLOCK   (1 << 5)
#define CAPSLOCK  (1 << 6)

void SetKbdStatus( int lockname, int status )
{
   char far* kbdstatus = (char far*)0x00400017UL;
   disable( );
   if ( status==ON )
      *kbdstatus |= (char)lockname;
      else
      *kbdstatus &= ~(char)lockname;
   enable( );
} /*--SetKbdStatus( )--------*/

int GetShiftFlags( void )
{
    asm{
        MOV AH, 2h;
        INT 16h;
      }
    return( _AL );
} /*--GetShiftFlags( )------*/

int main( void )
{
   SetKbdStatus( CAPSLOCK, ON );
   SetKbdStatus( NUMLOCK, ON );
   GetShiftFlags( );  /* Ignore the return value */
   return(0);
} /*--main( )--------*/
```

The function `SetKbdStatus( )` is used to change the status of the keyboard. The status lights, on recent keyboards may not reflect the change. In that case you may call INT 16, AH=2 (`GetShiftFlags( )`) to update the lights.

22.1.3.2 Changing keyboard status with ports

Port 64h(status port) is used for getting the status of keyboard controller.

Port 60h(keyboard controller data port) can be used as keyboard input buffer or keyboard output buffer. If bit1 of status port is 0, data should only be written. That is because, if bit1 of status port is 1, input buffer is full and no write access is allowed until the bit clears. If bit0 of status port is 1, data should only be read. This is because, if bit0 of status port is 1 the output buffer will be full (i.e., port 60h has data for system) and the bit (bit0) will be cleared after a read access.

To change the status of keyboard, we must send two consecutive byte values as commands to the data port. The first byte value must be EDh. The second byte contains the state to set LEDs.

Bitfields for LED status				
7653	2	1	0	Purpose
XXXX				reserved. should be set to 0
	X			Caps Lock LED on
		X		Num Lock LED on
			X	Scroll Lock LED on

```
#define KEYSTATUS       (0x64)
#define KEYDATA         (0x60)
#define LEDUPDATE       (0xED)

#define OB_FULL         (1 << 0)    /* output buffer full */
#define IB_FULL         (1 << 1)    /* input buffer full */
#define KEY_ACK         (0xFA)

/* bit masks to be sent */

#define SCROLLLOCK      (1 << 0)
#define NUMLOCK         (1 << 1)
#define CAPSLOCK        (1 << 2)

/*-----------------------------------------------
      SendKeyControl - Receives the command
            'cmd' and returns  1 for success */

int SendKeyControl( int cmd )
{
   int byte;
   do
    {
      byte = inportb( KEYSTATUS );
    } while ( byte & IB_FULL );
```

```
   outportb( KEYDATA, cmd );

   do
    {
      byte = inportb( KEYSTATUS );
    } while ( byte & OB_FULL );
   byte = inportb( KEYDATA );

   /* if byte is KEY_ACK, then success */
   return( ( byte == KEY_ACK ) );
} /*--SendKeyControl( )------------*/

int main( void )
{

    if ( SendKeyControl( LEDUPDATE ) )    /* tell keyboard next
                                  byte is LED bitmask */
            SendKeyControl( CAPSLOCK );   /* the LED bitmask */
    return(0);
} /*--main( )-----*/
```

22.1.4 Keyboard Interrupt

To get scan code of ASCII character of the key pressed, we can use the INT 16, AH=10h (Get Enhanced Keystroke). This function returns BIOS scan code in AH and ASCII character in AL register. If no keystroke is available, this function waits until one is placed in the keyboard buffer. The BIOS scan code is usually, but not always, the same as the hardware scan code processed by INT 09 or the one we get from Port 60h. It is the same for ASCII keystrokes and most unshifted special keys (F-keys, arrow keys, etc.), but differs for shifted special keys.

22.2 Activating the keys without pressing it!

We can ‘press’ the keys through programs. This technique is referred as “stuff keys” by programmers. We can stuff keys with BIOS interrupt 16h or with keyboard buffer. Usually stuff keys technique is used for cracking passwords and it is explained in “Illegal Codes” section..

22.2.1 Stuff keys using BIOS interrupt

BIOS interrupt 16h function 5h can be used to stuff keys. Usually all BIOS support this interrupt.

22.2.2 Stuff keys using keyboard buffer

We can also stuff keys using keyboard buffer. This is widely used for cracking passwords with brute force technique. The code below was actually by **Alexander Russell**. I have restructured it for the sake of clarity.

```
/*-------------------------------------------------------------
      Stuffkey.c
      stuff chars into the BIOS keyboard buffer then exit
 *----
 */

#include <string.h>
#include <dos.h>

/*-----------------------------------------------------
      Stuff - stuffs ch into BIOS keyboard buffer     */

void Stuff( char ch )
{
   unsigned far *kbufbegin;
   unsigned far *kbufend;
   unsigned far *kbuftail;
   unsigned far *kbuffer;

   /* set up buffer pointers */

   FP_SEG( kbufbegin ) = FP_SEG( kbufend ) = FP_SEG( kbuftail )
                    = FP_SEG( kbuffer ) = 0x40;
   FP_OFF( kbufbegin ) = 0x80;
   FP_OFF( kbufend ) = 0x82;
   FP_OFF( kbuftail ) = 0x1c;

   disable( );

   FP_OFF( kbuffer ) = *kbuftail;

   *kbuffer++ = ch;

   if ( FP_OFF( kbuffer ) >= *kbufend )
            FP_OFF( kbuffer ) = *kbufbegin;

   *kbuftail = FP_OFF( kbuffer );

   enable( );
} /*--Stuff( )---------------*/


int main( int argc, char *argv[] )
{
   short i, j;
```

```
   char  ch;
   char temp[200];
   if ( argc > 1 )
      for ( i=1; i < argc; ++i )
        {
            strcpy( temp, argv[i] );
            switch ( temp[0] )
            {
               case '0':
                      ch = atoi( temp );
                      Stuff( ch );
                      break;
               default:
                      for ( j=0; temp[j] != '"' && temp[j]; ++j )
                             Stuff( temp[j] );
            }
        }
    else
     {
       printf( "Use: STUFFKEY 027 013 a b \"hi there\"<ENTER>\n");
       printf( "Parms that start with zero are ascii codes\n" );
       printf("Generaly only useful called from inside a batch file\n");
     }
   return(0);
} /*--main( )----------*/
```

According to theory, keyboard buffer stores both ASCII and scan codes in alternate bytes. But the above code stuffs only ASCII code. So the success of the above code depends upon the reading program written in BIOS. For me the above code works fine. If it doesn't work for you, try to stuff scan code too and it should work.

22.3 Multiple key Input

The following program explains how to get multiple key input. This has many applications. One of them is Piano programming where we would press more than one key. In order to test this program, don't forget to press more than one key!

```
#define PRESSED   (1)
#define RELEASED  (0)
#define ESC       (1)
typedef int BOOLEAN;

char *Keys_Tbl[88] = {
     /* 1..8 */         "Escape", "1", "2", "3", "4",  "5",  "6",  "7",
     /* 9..15 */  "8",  "9",  "0",  "-",  "=",  "Backspace", "Tab",
     /* 16..25 */ "q", "w", "e", "r", "t", "y", "u", "i", "o", "p",
```

```
      /* 26..29 */ "[", "]", "Enter/KeypadEnter", "Left/RightCtrl",
      /* 30..39 */ "a", "s", "d", "f", "g", "h", "j", "k", "l", ";",
      /* 40..42 */ "'", "`", "LeftShift/PrintScreen",
      /* 43..45 */ "\\(101-keyOnly)/#(102-keyOnly)", "z", "x",
      /* 46..53 */ "c", "v", "b", "n", "m", ",", ".", "/",
      /* 54..55 */ "RightShift", "Keypad*/PrintScreen",
      /* 56..59 */ "Left/RightAlt", "Spacebar", "Caps Lock", "F1",
      /* 60..67 */ "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9",
      /* 68..70 */ "F10", "NumLock/Pause", "ScrollLock",
      /* 71..72 */ "Home/Keypad7", "UpArrow/Keypad8",
      /* 73..74 */ "PageUp/Keypad9", "Keypad-",
      /* 75..76 */ "LeftArrow/Keypad4", "Keypad5",
      /* 77..78 */ "RightArrow/Keypad6", "Keypad+",
      /* 79..80 */ "End/Keypad1", "DownArrow/Keypad2",
      /* 81..82 */ "PageDown/Keypad3", "Insert/Keypad0",
      /* 83..85 */ "Delete/Keypad.", "undefined", "undefined",
      /* 86..88 */ "\(102-keyOnly)", "F11", "F12"
                  };
BOOLEAN Key_Stat[88];
int main( void )
{
    int i, key;
    while( (key=inportb(0x60))!=ESC )
     {
      /* Store the status of keys... */
      if ( key<128 )
            Key_Stat[key-1] = PRESSED;
        else
            Key_Stat[key-1-128] = RELEASED;
      /* Now, show the status... */
      for ( i=0; i<88 ; ++i )
         if ( Key_Stat[i]==PRESSED )
            printf( "%s ", Keys_Tbl[i] );
      printf( "\n" );
     }
    return(0);
} /*--main( )----------*/
```

Exercises

1. Write `getch( )` and `kbhit( )` functions without using any interrupt. (Hint: use keyboard buffer)
2. Write a program that temporarily lock or freeze the system. (i.e. to lock keys)
3. Write a program to find out the size of the keyboard buffer.

4. Write a function `ASCII2Scan( )` that returns scan code for the given ASCII value using system resources i.e., don't pre-calculate the values. (It's really a tough job! Hint: you have to crack the driver file)
5. Write a "running lights" program using CAPSLOCK, NUMLOCK & SCROLLLOCK LEDs.

Suggested Projects

1. Write software to increase or decrease the size of the keyboard buffer.
2. Use stuff key techniques and interfacing techniques to input keys from other devices.